|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
|  | | |
| Домашнее задание № 3 | | |
| по дисциплине «Методы построения и анализа алгоритмов» | | |
|  | | |
| NP-полные задачи | | |
|  | | |
| Группа: | ПМИ-81 |
|  |  |
| Студент: | Кондратьев Игорь |
|  | Вариант - 5 |
|  |  |
| Преподаватели: | Щукин Георгий Анатольевич |
|  |  |
|  | | |
| Новосибирск | | |
| 2019 | | |

1. **Условие задачи**

Для решения NP-полной задачи из варианта требуется разработать (реализовать) два алгоритма:

1. алгоритм, решающий задачу методом полного перебора/комбинаторного поиска и гарантированно находящий оптимальное решение задачи. С большой вероятностью этот алгоритм будет иметь экспоненциальную временную сложность
2. алгоритм, находящий субоптимальное (приближенное) решение задачи, но имеющий полиномиальную временную сложностью. Классы алгоритмов, находящих субоптимальное решение: жадные алгоритмы, рандомизированные алгоритмы, эвристические алгоритмы и т.д.

Дано множество городов. Для каждой пары городов известна цена перелета из одного города в другой. Для заданного стартового города найти такой маршрут, который проходил бы ровно по одному разу через все остальные города, при этом суммарные затраты на перелеты были бы минимальны.

1. **Использованные алгоритмы:**

Оценка сложности алгоритма: Полный перебор – **O((n – 1)!),**

Жадный алгоритм – **O(n2)**

1. **Текст программы**

using System;

using System.Collections.Generic;

using System.Diagnostics;

namespace MPA\_HW3

{

public class Program

{

public static List<int> FindFullWayByBruteForce(int[,] graph, int startIndex, out int length)//поиск кратчайщего путем полного перебора

{

if(graph.Length <= 1)

{

throw new Exception("NonPositiveGrpahSize");

}

length = 0;

List<int> minWay = new List<int>();

int[] way = new int[graph.GetLength(0) - 1];

//way[0] = startIndex;

int k = 0;

if (startIndex == 0)

{

k = 1;

}

for (; k < graph.GetLength(0); k++)

{

if(k < startIndex)

{

way[k] = k;

}

else if(k > startIndex)

{

way[k - 1] = k;

}

}

int minCost = int.MaxValue;

int cost = 0;

bool isEnd = false;

do

{

cost = graph[startIndex, way[0]];

for (int i = 0; i < way.Length - 1; i++)

{

cost += graph[way[i], way[i + 1]];

}

if(cost < minCost)

{

minCost = cost;

minWay = new List<int>();

minWay.Add(startIndex);

for (int i = 0; i < way.Length; i++)

{

minWay.Add(way[i]);

}

}

isEnd = !NextPermutation(way);

Console.WriteLine("[{0}, {1}]", startIndex, string.Join(", ", way));

}

while (!isEnd);

length = minCost;

return minWay;

}

public static bool NextPermutation(int[] sequence)//следующая последовательность перестановки

{

int i = sequence.Length;

do

{

if (i < 2)

{

return false;

}

i--;

} while (sequence[i - 1] >= sequence[i]);

int j = sequence.Length;

while (i < j && sequence[i - 1] >= sequence[--j]);

SwapItems(sequence, i - 1, j);

j = sequence.Length;

while (i < --j)

{

SwapItems(sequence, i++, j);

}

return true;

void SwapItems(int[] array, int index\_0, int index\_1)

{

var item = array[index\_0];

array[index\_0] = array[index\_1];

array[index\_1] = item;

}

}

public static List<int> FindFullWayByGA(int[,] graph, int startIndex, out int length)//поиск кратчайщего жадным алгоритмом

{

if (graph.Length <= 1)

{

throw new Exception("NonPositiveGrpahSize");

}

length = 0;

List<int> way = new List<int>();

way.Add(startIndex);

int searchIndex = startIndex;

while (way.Count != graph.GetLength(0))

{

int minCost = int.MaxValue;

int minIndex = -1;

for (int i = 0; i < graph.GetLength(1); i++)

{

if (graph[searchIndex, i] < minCost && graph[searchIndex, i] != -1 && way.IndexOf(i) == -1)

{

minIndex = i;

minCost = graph[searchIndex, i];

}

}

way.Add(minIndex);

length += minCost;

searchIndex = minIndex;

}

return way;

}

public static int[,] GenerateGraph(int size)//генерация случайной сети городов

{

if(size <= 0)

{

throw new Exception("NonPositiveGrpahSize");

}

int[,] graph = new int[size, size];

Random random = new Random();

for (int i = 0; i < size; i++)

{

for (int j = 0; j < size; j++)

{

if(i == j)

{

graph[i, j] = -1;

}

else

{

graph[i, j] = random.Next(1, 40);

}

}

}

return graph;

}

public static void Main(string[] args)

{

Console.WriteLine("Enter start index:");

int startIndex = int.Parse(Console.ReadLine());

Console.WriteLine("Enter graph size:");

int graphSize = int.Parse(Console.ReadLine());

int[,] graph = GenerateGraph(graphSize);

//Brute force

Stopwatch sw = new Stopwatch();

sw.Start();

int lengthBF;

List<int> way = FindFullWayByBruteForce(graph, startIndex, out lengthBF);

sw.Stop();

Console.WriteLine("Elapsed={0}", sw.Elapsed.TotalSeconds);

Console.WriteLine($"Min way length by brute force: {lengthBF}");

Console.WriteLine("[{0}]", string.Join(", ", way));

//MBB

sw = new Stopwatch();

sw.Start();

int lengthGA;

way = FindFullWayByGA(graph, startIndex, out lengthGA);

sw.Stop();

Console.WriteLine("Elapsed={0}", sw.Elapsed.TotalSeconds);

Console.WriteLine($"Min way length by gready method: {lengthGA}");

Console.WriteLine("[{0}]", string.Join(", ", way));

Console.ReadKey();

}

}

}

**4.Текст юнит тестов**

using System;

using System.Collections.Generic;

using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace MPA\_HW3.Tests

{

[TestClass]

public class ProgramTests

{

[TestMethod]

public void GenerateGraph\_NonPositiveSize\_Exception()

{

//arrange

string expected = "NonPositiveGrpahSize";

//act

Exception exception = Assert.ThrowsException<Exception>(() => Program.GenerateGraph(-1));

string actual = exception.Message;

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void GenerateGraph\_PositiveSize\_Generated()

{

//arrange

int expected = Program.GenerateGraph(5).GetLength(0);

//act

int actual = 5;

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void NextPermutation\_Size3\_Done()

{

//arrange

int[] sequence = { 1, 2, 3};

int[,] expected =

{

{1, 2, 3 },

{1, 3, 2 },

{2, 1, 3 },

{2, 3, 1 },

{3, 1, 2 },

{3, 2, 1 }

};

//act

int[,] actual = new int[6, 3];// 6 = 3 \* 2 \* 1

int i = 0;

for (int j = 0; j < sequence.Length; j++)

{

actual[i, j] = sequence[j];

}

i++;

while (Program.NextPermutation(sequence))

{

for (int j = 0; j < sequence.Length; j++)

{

actual[i, j] = sequence[j];

}

i++;

}

//assert

for (i = 0; i < expected.GetLength(0); i++)

{

for (int j = 0; j < expected.GetLength(1); j++)

{

Assert.AreEqual(expected[i, j], actual[i, j]);

}

}

}

[TestMethod]

public void NextPermutation\_Empty\_FalseReturn()

{

//arrange

int[] sequence = { };

bool expected = false;

//act

bool actual = Program.NextPermutation(sequence);

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void FindFullWayByBruteForce\_EmptyOr1Size\_Exception()

{

//arrange

int[,] graph = { };

int startIndex = 0;//no matter

string expected = "NonPositiveGrpahSize";

//act

int actualLenght;

Exception exception = Assert.ThrowsException<Exception>(() => Program.FindFullWayByBruteForce(graph, startIndex, out actualLenght));

string actual = exception.Message;

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void FindFullWayByBruteForce\_3SizeGraph\_Found()

{

//arrange

int[,] graph =

{

{-1, 22, 32 },

{11, -1, 12 },

{20, 10, -1 }

};

int startIndex = 0;

List<int> expected = new List<int>();

expected.Add(startIndex);

expected.Add(1);

expected.Add(2);

int expectedLenght = 34;//22 + 12

//act

int actualLenght;

List<int> actual = Program.FindFullWayByBruteForce(graph, startIndex, out actualLenght);

//assert

CollectionAssert.AreEqual(expected, actual);

Assert.AreEqual(expectedLenght, actualLenght);

}

[TestMethod]

public void FindFullWayByBruteForce\_4SizeGraph\_Found()

{

//arrange

int[,] graph =

{

{-1, 5, 11, 9},

{10, -1, 8, 7 },

{7, 14, -1, 8 },

{12, 6, 15, -1 }

};

int startIndex = 0;

List<int> expected = new List<int>();

expected.Add(startIndex);

expected.Add(1);

expected.Add(2);

expected.Add(3);

int expectedLenght = 21;//5 + 8 + 8

//act

int actualLenght;

List<int> actual = Program.FindFullWayByBruteForce(graph, startIndex, out actualLenght);

//assert

CollectionAssert.AreEqual(expected, actual);

Assert.AreEqual(expectedLenght, actualLenght);

}

[TestMethod]

public void FindFullWayByGA\_EmptyOr1Size\_Exception()

{

//arrange

int[,] graph = { };

int startIndex = 0;//no matter

string expected = "NonPositiveGrpahSize";

//act

int actualLenght;

Exception exception = Assert.ThrowsException<Exception>(() => Program.FindFullWayByGA(graph, startIndex, out actualLenght));

string actual = exception.Message;

//assert

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void FindFullWayByGA\_3SizeGraph\_Found()

{

//arrange

int[,] graph =

{

{-1, 22, 32 },

{11, -1, 12 },

{20, 10, -1 }

};

int startIndex = 0;

List<int> expected = new List<int>();

expected.Add(startIndex);

expected.Add(1);

expected.Add(2);

int expectedLenght = 34;//22 + 12

//act

int actualLenght;

List<int> actual = Program.FindFullWayByGA(graph, startIndex, out actualLenght);

//assert

CollectionAssert.AreEqual(expected, actual);

Assert.AreEqual(expectedLenght, actualLenght);

}

[TestMethod]

public void FindFullWayByGA\_4SizeGraph\_Found()

{

//arrange

int[,] graph =

{

{-1, 5, 11, 9},

{10, -1, 8, 7 },

{7, 14, -1, 8 },

{12, 6, 15, -1 }

};

int startIndex = 0;

List<int> expected = new List<int>();

expected.Add(startIndex);

expected.Add(1);

expected.Add(3);

expected.Add(2);

int expectedLenght = 27;//5 + 7 + 15

//act

int actualLenght;

List<int> actual = Program.FindFullWayByGA(graph, startIndex, out actualLenght);

//assert

CollectionAssert.AreEqual(expected, actual);

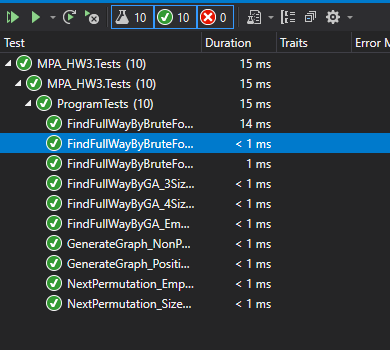
Assert.AreEqual(expectedLenght, actualLenght);

}

}

}

**5. Результаты тестов**

****

**6.Графики зависимости времени от количества городов**

**1) Алгоритм полного перебора**

Кол-во городов

Сек.

**2) Жадный алгоритм**

Кол-во городов

Сек.

Точность жадного алгоритма:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Кол-во городов | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |  |
| Полный перебор | 41 | 25 | 67 | 55 | 52 | 34 | 53 | 43 | цена |
| Жадный алгоритм | 41 | 31 | 71 | 68 | 76 | 49 | 67 | 49 | цена |

**7.Вывод**

В ходе выполнения работы была выполнена:

* 1. Реализация алгоритма полного перебора для данной задачи.
  2. Реализация жадного алгоритма для данной задачи.
  3. Проведено тестирование на проверку корректности реализации алгоритмов и программы.
  4. Проведено исследование времени работы алгоритма на данных разного размера.
  5. Проведено исследование точности работы жадного алгоритма относительно полного перебора

Результаты:

1. Тестирование работает корректно.
2. Алгоритм полного перебора имеет сложность – **O((n – 1)!)**, а жадный алгоритм – **O(n2)**
3. Полный перебор дает точный результат, тогда как жадный алгоритм дает приближенный результат.